session cluster api

API Specification

Table of Contents

Overview 2

Addressing 2

handshake details 2

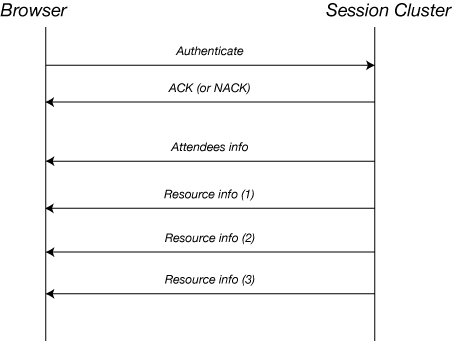
Authentication 3

Response 3

Attendees information 4

Resource information 4

# Overview



The above diagram shows the initial flow of user joining a session. All messages are transported over WebSockets.

# Addressing

All addresses in the communication protocol have the following addressing format:

Class[:instance].sub-class[:instance]...

As an example, a message to a user **padmasambhava’s** AV module would be addressed as:

user:padmasambhava.av

A address for the authentication module within the session cluster controller would look like this:

controller.auth

# handshake details

## Authentication

{

"v": 1,

"type": "req",

"to": "controller.auth",

"from": "user:-not-yet-authenticated-",

"msg": <auth-encrypted-string>,

"seq": 0

}

The *auth-encrypted-string* is the verbatim, encrypted string handed down by the Authentication Gateway to the VC Application sub-system. Currently it is just a URI encoded string, containing the following information (in the specified format):

{ id : <some-user-id>, displayName : <display-name> }

### Response

Since this is a request, it must receive an acknowledgement. If the authentication succeeds, an ‘ACK’ is returned. A sample is shown below:

{

"v": 1,

"type": "ack",

"seq": 0,

"to": "user:-not-yet-authenticated-",

"from": "controller.auth",

"msg": {

"status": "ok",

"data": {

"id": "a",

"displayName": "Avinash",

"history": [

{

"joined": "2016-02-25T09:33:45.465Z"

},

{

"removed": "2016-02-25T10:20:25.482Z"

},

{

"joined": "2016-02-25T10:28:59.112Z"

},

{

"removed": "2016-02-25T10:29:12.065Z"

},

{

"joined": "2016-02-25T10:29:13.111Z"

}

],

"vc\_id": "JGr1M2",

"nickname": "distracted williams"

}

}

}

An ‘ACK’ will always be of the following form:

{

"v": 1,

"type": "ack",

"seq": <sequence number of the request>,

"to": <to-address>,

"from": <from-address>,

"msg": {

"status": “ok” | “not-ok”,

"data": {

any custom data or empty

}

}

}

## Attendees information

Once the class is started, the user is automatically sent the attendees information, even if it is has no other attendees at the moment. The transmission of this information indicates that the class has begun. The following is an example of a (no attendees) info packet. Note that this is an INFO message and does not require any ACK from it’s recipient (in our case, the user’s framework). Note, also the “info\_id” field – this helps the recipient framework from routing this message to the correct module – in this case, this message will be routed to the “attendees” module.

{

"v": 1,

"type": "info",

"seq": 7,

"to": "user:JGr1M2.framework",

"from": "controller",

"msg": {

"info\_id": "session-info",

"info": {

"attendees": []

}

}

}

## Resource information

Once the class begins or when a new attendee joins an already started class, the process of resource initialization per user begins. Each resource may take it’s own time to initialize. An INFO packet is sent to the newly joined user upon the completion of each resource initialization routine. The information sent in these packets is essential for the client side resource to begin functioning. For example, for the chat resource this INFO packet may contain the name of the (dynamically created) chat root and so on.

{

"v": 1,

"type": "info",

"seq": 8,

"to": "user:JGr1M2.framework",

"from": "user-controller",

"msg": {

"info\_id": "resource-init",

"info": {

"status": "ok",

"name": "av-tokbox-v2",

"info": {

>> every thing inside this is a per-resource, free-flowing format message

>> which likely, only the resource itself will understand. It is shown here

>> only as an example.

"sessionid": "1\_MX40NTQ1Nzc4Mn5-MTQ1NjM5NzY5MjA5N35uYW1RTlprdkIrWVdaZ25qR0VHQm1FUlJ-fg",

"token": "T1==cGFydG5lcl9pZD00NTQ1Nzc4MiZzaWc9ZTFhZmU0MjBmNWFhZWNjZGZhNzliMTdmODhjY2FlZThmZjQzY2JkNTpzZXNzaW9uX2lkPTFfTVg0ME5UUTFOemM0TW41LU1UUTFOak01TnpZNU1qQTVOMzV1WVcxUlRscHJka0lyV1ZkYVoyNXFSMFZIUW0xRlVsSi1mZyZjcmVhdGVfdGltZT0xNDU2Mzk2MTUzJm5vbmNlPTAuNzE1MzU4Njg4MjYzMjIyNiZyb2xlPW1vZGVyYXRvciZleHBpcmVfdGltZT0xNDU2NTY4OTUzJmNvbm5lY3Rpb25fZGF0YT1KR3IxTTI=",

"key": "45457782",

"classid": null,

"username": "JGr1M2",

"authid": null,

"chromelocalextensionid": "hmlfjgegnbjjhaacghlfekifkagkmbce",

"chromeextensionid": "cofnnopnhjmpoomoholnofbneelimjdm",

"inlinechromeextinstall": false

}

}

}

}

# Keep Alives

The onus of keeping the websocket connection alive is on the client. It is required to send a keep alive ‘ping’ message approximately every 10 seconds (for now). The default timeout of websockets (which could be implementation dependent) appears to be 3 mins.

## The ‘Ping’ message

{

v: 1,

seq: <seq-no>,

type:"ping"

}

## The ‘PONG’ response message

{

v: 1,

seq: <seq-no>, #same as the ping messge

type:"pong"

}